CHAPTER 7: COMPUTATIONAL THINKING AND PROBLEM-SOLVING

7.1 RECURSION

7.1.1 Essential Features

Base Case:

- Stopping condition
- Prevents infinite recursion

Recursive Case:

- Calls itself
- Changes state toward base case

Unwinding:

- When base case reached
- Returns through call stack
- Calculates final result

7.1.2 Advantages and Disadvantages

Advantages	Disadvantages
Simpler, more natural solutions	Less efficient (time and space)
Elegant for tree-like problems	Stack overflow risk
Reduced code complexity	Hard to debug

7.1.3 How Compilers Handle Recursion

Stack Usage:

1. Before call: Save current state (registers, local variables) onto stack
2. Make recursive call
3. During return: Restore state from stack

Unwinding:

- When base case met
- Values popped from stack in reverse order
- Previous call states restored

7.1.4 Examples

Factorial:

<PYTHON>

```
def factorial(n):
 if n <= 1:
 return 1
 else:
 return n * factorial(n-1)
```

Fibonacci:

<PYTHON>

```
def fibonacci(n):
 if n <= 1:
 return n
 else:
 return fibonacci(n-1) + fibonacci(n-2)
```

7.2 ALGORITHM COMPLEXITY

7.2.1 Big O Notation

Notation	Name	Example
O(1)	Constant	Array index access
O(log n)	Logarithmic	Binary search
O(n)	Linear	Linear search
O(n log n)	Linearithmic	Merge sort
$O(n^2)$	Quadratic	Bubble sort

7.2.2 Comparing Algorithms

Time Complexity:

- How execution time grows with input size

Space Complexity:

- How memory usage grows with input size

Revision #1
Created 2026-03-16 12:17:12 UTC by Samuel Lee
Updated 2026-03-16 12:17:24 UTC by Samuel Lee